System.Text.Json

September 4, 2026

System.Text.Json

Kind: Service

Source: `atloria-monorepo/libs/parser-core/src/dotnet-bridge/Atloria.CSharpParser/Atloria.CSharpParser.csproj`

NuGet package dependency

`System.Text.Json` is a NuGet dependency used by the `Atloria.CSharpParser` .NET bridge to serialize parser results and deserialize structured requests. It provides the JSON boundary between TypeScript consumers in the monorepo and the C#-based parsing implementation.

Diagram

```
sequenceDiagram
    participant TS as TypeScript Consumer
    participant Bridge as Atloria.CSharpParser
    participant JSON as System.Text.Json
    participant Roslyn as C# Parser

    TS->>Bridge: Send JSON parser request
    Bridge->>JSON: Deserialize request payload
    JSON-->>Bridge: Typed .NET model
    Bridge->>Roslyn: Parse C# source
    Roslyn-->>Bridge: Syntax/result data
    Bridge->>JSON: Serialize response
    JSON-->>TS: JSON parser result
```

Usage

```
type ParserRequest = {
  source: string;
  fileName: string;
};
```

```
const request: ParserRequest = {
  fileName: "Example.cs",
  source: `
    namespace Demo;

    public class Greeter
    {
      public string SayHello() => "Hello";
    }
  `,
};

// Send JSON to the .NET parser bridge through its configured transport,
// such as a CLI process, IPC channel, or HTTP endpoint.
const response = await fetch("http://localhost:3000/parse/csharp", {
  method: "POST",
  headers: {
    "content-type": "application/json",
  },
  body: JSON.stringify(request),
});

if (!response.ok) {
  throw new Error(`C# parser request failed: ${response.statusText}`);
}

const result = await response.json();
console.log(result);
```

AI Coding Instructions

- Use `System.Text.Json` for all request and response serialization at the .NET bridge boundary; avoid introducing a second JSON library unless required.
- Keep JSON request and response models explicit and stable, since TypeScript consumers depend on their serialized property names and shapes.
- Configure serialization options consistently across bridge entry points, especially property naming policies and null-value handling.
- Treat malformed JSON as an input validation error and return structured diagnostics rather than allowing unhandled deserialization exceptions.
- When changing C# result models, update the corresponding TypeScript types and integration tests that consume the serialized parser output.

Referenced By

- `Atloria.CSharpParser` (DEPENDS_ON)